Army Research Laboratory

Automatic Stripe Analysis Tool

by Justin R. Bickford

ARL-TR-6469 May 2013

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-6460 **May 2013**

Automatic Stripe Analysis Tool

Justin R. Bickford
Sensors and Electron Devices Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. **REPORT DATE** *(DD-MM-YYYY)* May 2031	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** November 2012 to March 2013
4. TITLE AND SUBTITLE Automatic Stripe Analysis Tool		**5a. CONTRACT NUMBER** HR0011-08-09-0001
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Justin R. Bickford		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-SEE-E 2800 Powder Mill Road Adelphi, MD 20783-1197		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-6469
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA UNIC 675 N Randolph St Arlington VA 22203		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report discusses the design and implementation of an automatic stripe analysis application for use in metrology. It has been implemented in Mathworks' Matlab scripting environment and wrapped in an easy-to-use graphical user interface (GUI). The algorithm is robust against common image artifacts and can analyze several image formats. The automatic nature of the script removes human error and allows large numbers of images to be analyzed quickly with high precision. The stripe width precision is generally better than the resolution of the input image.

15. SUBJECT TERMS
Optical Microscope, SEM, Stripe Analysis, Metrology

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Justin R. Bickford
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	30	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-5127

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

This work was performed in support of the Defense Advanced Research Projects Agency's (DARPA) Ultraperformance Nanophotonic Intrachip Communication (UNIC) program contract # HR0011-08-09-0001.

INTENTIONALLY LEFT BLANK.

1. Introduction

This automatic stripe analysis tool was initially developed to characterize the e-beam patterning of hydrogen silsesquioxane (HSQ) on a silicon on insulator (SOI) substrate, but it is applicable to a wide array of metrology applications. In order to produce HSQ resist patterns that are faithful to their as-drawn feature geometries, we must account for the effective point spread function (PSF) of the electronic beam (e-beam). The PSF is a function of the e-beam backscatter from the substrate and other proximity effects associated with resist development. The correction of the PSF is called proximity effect correction (PEC). The first step in measuring the PSF is to analyze the linewidths of several line and space patterns written at different doses in a dose matrix (figure 1). Often, these types of dose matrices consist of a large number of test features (~100) and, to correctly resolve the PSF, the linewidths of the patterned features must be resolved to an accuracy of a few nanometers, despite the ~10-nm resolution of the scanning electron microscopy (SEM) images. This tool provides precision far beyond traditional human picked feature width metrology.

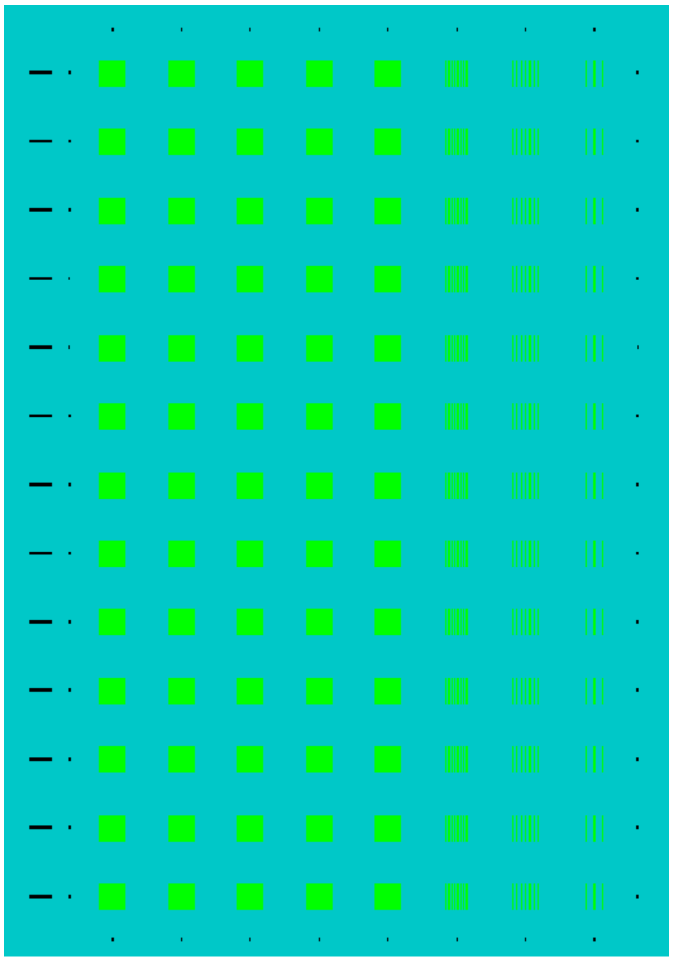


Figure 1. Line-space dose matrix example.

The algorithm is split into several steps including image calibration, automatic rotation, edge finding, and stripe analysis. The script was packaged with a graphical user interface (GUI) for ease of use. All examples are of SEM images; however, the script is applicable to optical microscope and macroscopic images. The program was written in Mathworks' Matlab scripting environment.

2. Methods, Assumptions, and Procedures

2.1 Image Calibration

In order to produce measurements applicable to the real world, the pixels per meter must be calibrated. This calibration only needs to be preformed once for every magnification. The GUI allows the user to draw a line across a known feature size and then enter its physical size in meters to establish a calibration (figure 2). As the line end placement accuracy is limited by the resolution of the image, longer lines yield more precise calibrations.

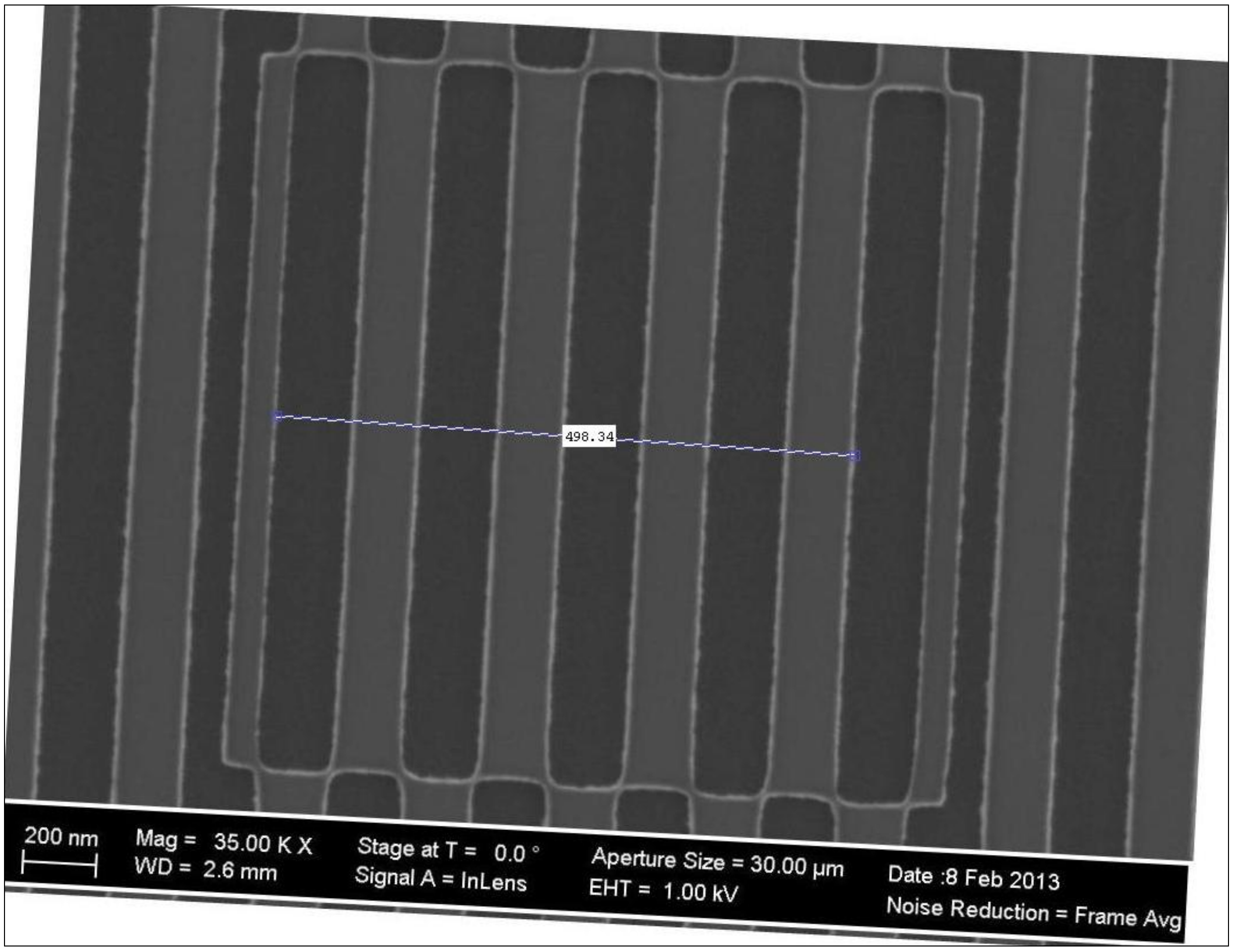


Figure 2. Calibration line example.

2.2 Automatic Rotation

When analyzing an image, the first task is to choose a region of interest (ROI) to analyze, as shown in figure 3a. Older revisions of this algorithm allowed the user to select multiple stripes. However, those older revisions analyzed the stripes with a more primitive method and were susceptible to image compression artifacts and only worked on distinctly bright stripes. The latest revision uses an edge finding method and can analyze single stripes that are bright, dark, or of equal intensity (as long as their edges are distinct). The example starting image shown in figure 3a shows a stripe being selected that is approximately 3° off from vertical.

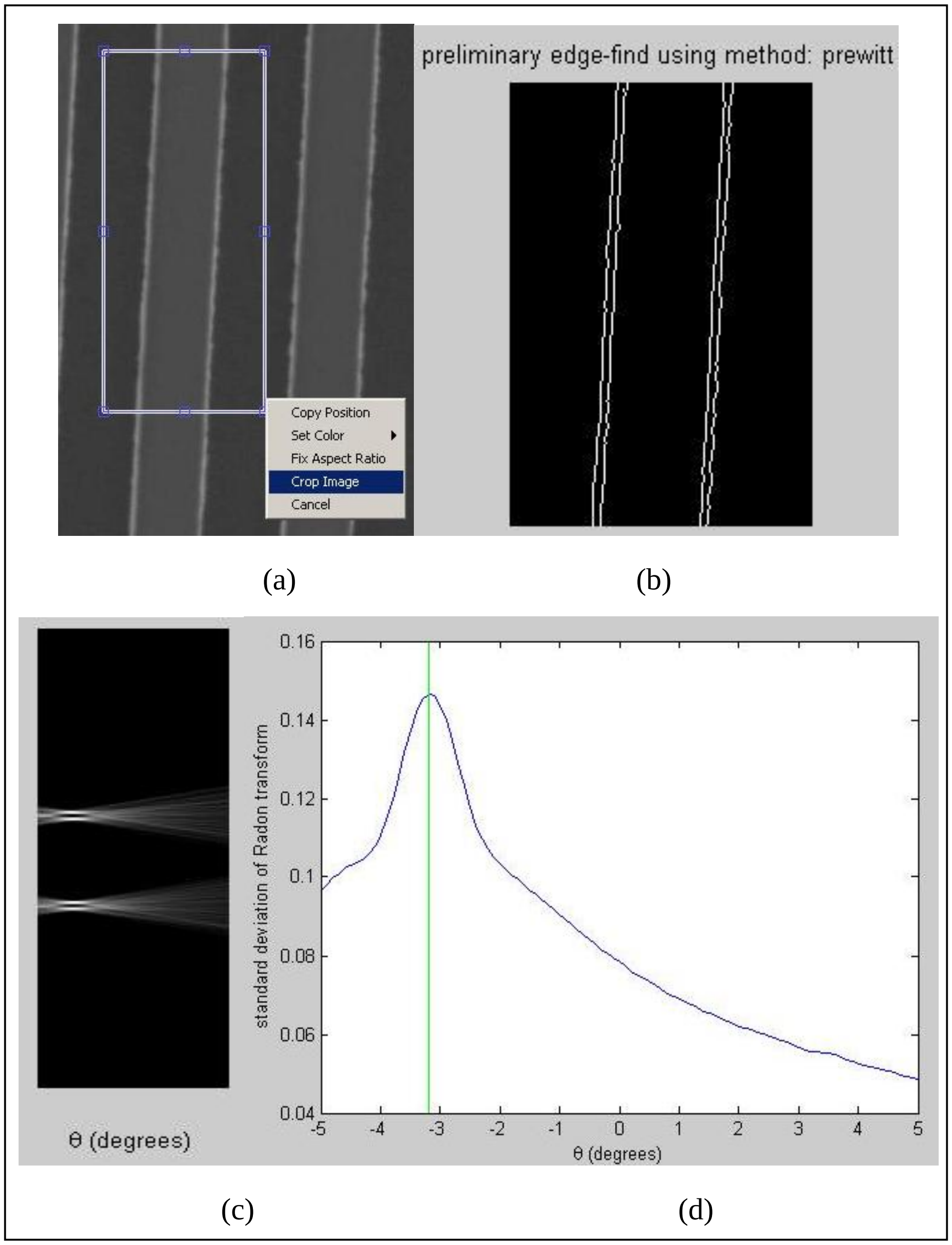
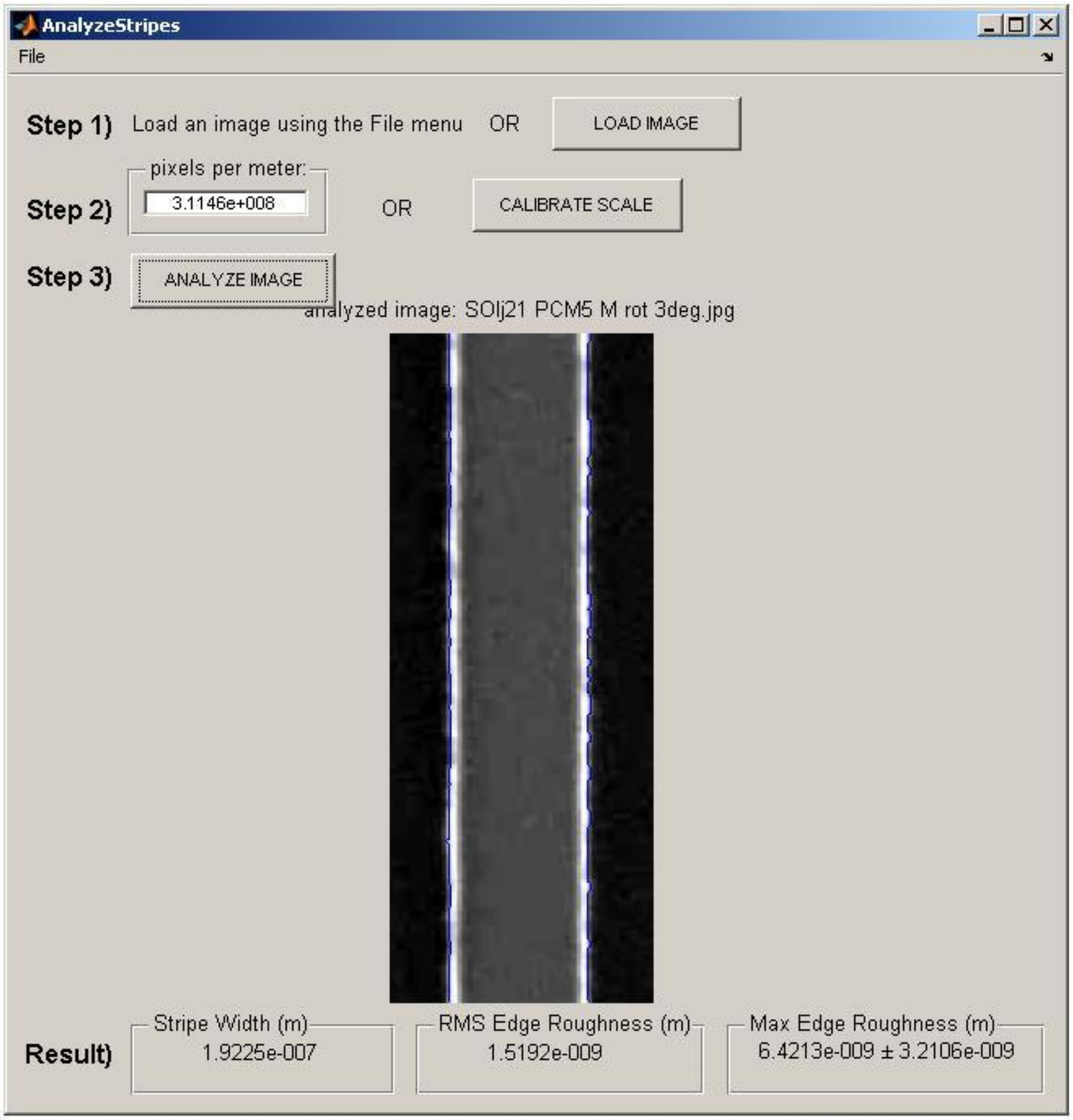


Figure 3. (a) ROI selection, (b) binary edge image, (c) Radon transform, and (d) standard deviation of a Radon transform.

After selecting the ROI, the remaining tasks are done automatically. After the ROI is defined, the next task is to preliminarily find edges. Several methods exist, though I found the most reliable was the Prewitt (*1*) method. The Prewitt edge finding method convolves the original intensity image with two 3x3 pixel filters, one that approximates the vertical gradient and one that approximates the horizontal gradient. The two resulting gradient maps are added in quadrature to approximate the gradient magnitude. This method is performed as a function in Matlab. The resulting image is actually a thresholded version (binary valued image) of the gradient magnitude image, where all gradient values above a threshold are considered edges and the remaining area is not. The result of performing the Prewitt edge detection on the ROI of figure 3a is shown in figure 3b.

In order to determine the stripe width, the rotation angle must be found. Once found, the original edges may be analyzed with the rotation angle or the original image may be rotated and the edges found again. It is easier and faster to perform the later. To find the rotation angle, I chose to perform a Radon transform (*2*) of the binary edge image. The Radon transform integrates the image pixel values along a given direction and then repeats this process for several different directions. It is typically performed over a full 180° set of rotation angles, but I have chosen to implement it over a high resolution subset extending from –5° to +5° in 0.1° increments. The result of performing the Radon transform of figure 3b is shown in figure 3c. It is less computationally intensive to perform the Radon transform on the binary edge image than it is to perform several trial rotations of the original image, and it yields a more accurate result than performing several trial rotations of the preliminary binary edge image. I have assumed, as is often the case, that the stripes will be either nearly vertical or nearly horizontal in the starting image. The ±5° Radon transform subset illustrated above is only meaningful for nearly vertical stripes. The above discussion is written to elucidate the concept; the script actually performs a pre-alignment check step that determines whether the binary edge image has nearly vertical or nearly horizontal edges and performs the Radon transform over either –5° to +5° or 85° to 95° appropriately. This pre-alignment check is done by averaging the binary edge image in the horizontal and vertical directions, and then measuring the standard deviation of both results. When the edges are nearly vertical, the vertical average yields a higher standard deviation and vice versa. This pre-alignment check is faster than performing two high-resolution Radon transforms.

The standard deviation of the Radon transform generates a plot with a peak at the ideal orientation angle. The standard deviation plot of the Radon transform of figure 3c is shown in figure 3d. This rotation angle is used to perform a bilinear rotation transform of the ROI image, resulting in a new ROI image with stripes that are vertical to within <0.05°. This rotated image is then automatically cropped to remove any border pixels resulting from the image rotation. A rotated and cropped ROI image is shown within the GUI of figure 4. After rotation, the same Prewitt filter is run on the rotated image, as shown in figure 5.



pixels per meter: 3.1146e+008

analyzed image: SOIj21 PCM5 M rot 3deg.jpg

Stripe Width (m): 1.9225e-007

RMS Edge Roughness (m): 1.5192e-009

Max Edge Roughness (m): 6.4213e-009 ± 3.2106e-009

Figure 4. GUI showing analyzed image output.

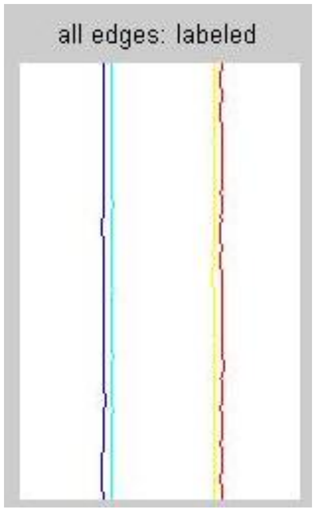


Figure 5. Automatically labeled edges.

2.3 Stripe Analysis

Note that (as is common in SEM images) the feature edges of figure 3a are highlighted due to electron charging and are detected as multiple edges. The first task in analyzing the stripes is to categorize and label each edge. This is done by a Matlab function that looks for connectedness among the edge lines. The resulting labeled edges are shown as colored lines in figure 5.

There are three cases of errors that occasionally occur and must be addressed. The first case is of edges that are not associated with the stripe geometry. These types of errors typically stem from image compression artifacts or particles in the field areas of the image. To combat this error, lines that extend <10% of the ROI image height are removed. The assumption here is that all artifacts or particles do not extend across a significant portion of the ROI image height.

The second error case is that of non-contiguous edges. To combat this, a search is performed across every edge line whereby any set of edges that are within the horizontal standard deviation of each other are relabeled as being part of the same edge.

The third error case arises when adjacent edge lines are connected resulting in labeled edges that contain multiple edge segments. The solution to this error is combined with the overall goal of the stripe analysis, which is to choose only the outermost edge pixels. This process results in two labeled edges that represent the exterior boarder of the stripe, whether it is a bright stripe, dark stripe, or of equal intensity (assuming their edges are distinct). The average distance between these exterior edges is the stripe width.

Often it is important to analyze the edges themselves to gauge edge roughness. The script averages the root mean square (RMS) edge roughness of each exterior line and outputs the result. In certain situations, the peak-to-peak edge roughness is more meaningful than the RMS edge roughness, so the script also outputs this value.

2.4 GUI Design

The script was wrapped with a GUI for ease of use by the end user, as shown in figure 4. This was done so that anyone in the organization that had proper Matlab licensing could use this utility without having to understand how it works. The GUI was designed to be very intuitive and efficient. The analysis process flow is split into three simple user tasks.

The first step is to load a starting image using a file browser similar to Windows. Once selected, the second step is either to enter a pixels-per-meter calibration value or click the calibrate scale button. The calibrate scale button invokes the calibration window shown in figure 2. The third step is to automatically perform the image rotation and stripe analysis. The stripe width and edge roughness values are output in the results area at the bottom of the GUI.

For ease of use, the calibration value remains until it is changed; therefore, multiple images may be loaded and analyzed within a single invocation of the GUI. Also, all relevant information about the analyzed image is logged in an output stream of text, including filepath, filename, pixels per meter, stripe width, RMS edge roughness, and peak-to-peak edge roughness.

3. Conclusions

The process of automatically analyzing striped features in images for metrology was discussed. The algorithm was implemented in Mathworks' Matlab scripting environment and wrapped with a GUI for ease of use by end users. The algorithm has been tested by several users and found to be robust to common image artifacts. The GUI outputs the average stripe with and edge roughness values. Note that the precision of the average stripe width is defined as the RMS edge roughness and that this precision is below the image resolution when ROI stripe selections are more than a few pixels in height. The longer the SOI, the more accurate the average stripe width result will be. By removing human error from the analysis, large numbers of images can be quickly analyzed with high precision.

The script itself is included in the appendix for interested readers. This script includes the GUI function calls; however, the associated GUI figure window is not text based and cannot be included with this report. Please contact me directly if you wish to investigate the GUI further or if would like to use the application.

4. References

1. Prewitt, J.M.S. *Object Enhancement and Extraction in Picture Processing and Psychopictorics*. Academic Press, 1970.
2. Radon, J. Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten, Berichte über die Verhandlungen der Königlich-Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse [Reports on the proceedings of the Royal Saxonian Academy of Sciences at Leipzig, mathematical and physical section] (Leipzig: Teubner) (69): 262–277; Translation: Radon, J.; Parks, P.C. (translator) (1986), On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging* **1917**, *5* (4), 170–176.

INTENTIONALLY LEFT BLANK.

Appendix. Source Code

The following is the Matlab script source code including the GUI portions.

```
function varargout = analyzeSEMstripesGUIv32(varargin)
% This script attempts to fix the inability of analyzeSEMstripes22.m
to
% properly analyze SOIj28 A1.tif. This script now finds the
stripewidth
% of any selection, whether they be bright or dark stripes depending
on how
% the ROI was selected. This version allows only one stripe to be
% selected. Image filetypes supported:
% (*.bmp,*.gif,*.jpg,*.pbm,*.pcx,*.pgm,*.png,*.ppm,*.ras,*.tif)
%
% v3.2 Changed file menu load image shortcut to ctrl-f instead of
ctrl-l.
% Added a GUI button to load images. Fixed a bug when you cancel the
load
% image file browser. Allowed the GUI to be rescalable. And added a
max
% edge span output to aid in roughness analysis.

% Last Modified by GUIDE v2.5 07-Mar-2013 07:36:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
 'gui_Singleton', gui_Singleton, ...
 'gui_OpeningFcn', @analyzeSEMstripesGUIv32_OpeningFcn, ...
 'gui_OutputFcn', @analyzeSEMstripesGUIv32_OutputFcn, ...
 'gui_LayoutFcn', [] , ...
 'gui_Callback', []);
if nargin && ischar(varargin{1})
 gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
 [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
 gui_mainfcn(gui_State, varargin{:});
end

format short eng
warning('off','Images:initSize:adjustingMag'); % shuts off image too
large warning
% End initialization code - DO NOT EDIT
% --- Executes just before analyzeSEMstripesGUIv32 is made visible.
function analyzeSEMstripesGUIv32_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to analyzeSEMstripesGUIv32 (see
VARARGIN)

% Choose default command line output for analyzeSEMstripesGUIv32
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using analyzeSEMstripesGUIv32.
if strcmp(get(hObject,'Visible'),'off')
 cla;
end

global filepath
filepath= pwd;


% UIWAIT makes analyzeSEMstripesGUIv32 wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = analyzeSEMstripesGUIv32_OutputFcn(hObject,
eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

global I pixelspermeter
```

```
caldistance_meters= 1600e-9;


%% calibrate pixelspermeter
%figure('Name','Click on two points, a known distance apart.');
figure('Name','Adjust the line''s endpoints to a known distance apart,
then press any key.');
imshow(I,[]);
hh= imdistline(gca);

% wait for a key to be pressed (not a mouse button)
w= waitforbuttonpress;
while w==0
 w= waitforbuttonpress;
end

api= iptgetapi(hh);
caldistance_pixels = api.getDistance(); % returns the line's distance
in pixels

prompt= {'Enter distance in meters:'};
dlg_title= '';
num_lines= 1;
def= {num2str(caldistance_meters,'%3.5g')};
answer= inputdlg(prompt,dlg_title,num_lines,def); % answer is a cell
array of strings
caldistance_meters= str2num(answer{1});
pixelspermeter= caldistance_pixels/caldistance_meters

close(gcf)

set(handles.edit2, 'String', num2str(pixelspermeter,'%3.5g'));



% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

global I pixelspermeter filename

%% crop the original image
figure('Name','Drag a window across the ROI in the image and right-
click to crop.');
J= imcrop(I); % prompts for cropping window and outputs the resulting
image
close(gcf)
J= imadjust(J);
```

```
%% find edges for rotation
method= 'prewitt';
Jbinmask= edge(J,method);
%figure, imshow(Jbinmask), title(['preliminary edge-find using method:
', method])


%% rotate image automatically
meanv= mean(J,1);
meanh= mean(J,2);
if std(meanv)>std(meanh) % if stripes are vertical
 theta = -5:0.1:5; % generates a reasonable set of misalignments to
choose from
 [R,xp] = radon(Jbinmask,theta); % performs Radon transform
 R= R/max(max(R));
 stdR= std(R,0,1); % finds the most probable stripe orientation
 maxstdR= max(stdR);
 rotationangle= mean(theta(stdR>maxstdR*.99));   % selects the most
probable stripe rotation angle, this one corrects for any near-ties
that may occur
 %rotationangle= theta(stdR==maxstdR);% selects the most probable
stripe rotation angle
 [r,c]= size(J); % necissary to prevent errors when croping below
else % if stripes are horizontal
 theta = 85:0.1:95; % generates a reasonable set of misalignments to
choose from
 [R,xp] = radon(Jbinmask,theta); % performs Radon transform
 R= R/max(max(R));
 stdR= std(R,0,1); % finds the most probable stripe orientation
 maxstdR= max(stdR);
 rotationangle= mean(theta(stdR>maxstdR*0.99));  % selects the most
probable stripe rotation angle, this one corrects for any near-ties
that may occur
 %rotationangle= theta(stdR==maxstdR);% selects the most probable
stripe rotation angle
 [r,c]= size(J'); % necissary to prevent errors when croping below
end

J= imrotate(J,-rotationangle,'bilinear','loose'); % rotates the
cropped image to vertical, very narrow sliver-like input J's yield
J=[] for some reason?
[r2,c2]= size(J);
dr= r2-r+1; % the +1 is necissary
dc= c2-c+1; % the +1 is necissary
J= J(dr:r2-dr,dc:c2-dc); % crops the image tightly to omit all
unwanted rotation data
[r,c]= size(J);
axes(handles.axes1);
imshow(J,[]); title(['analyzed image:
',filename],'Interpreter','none')
```

```
%% find the best edges
Jedge= edge(J,'prewitt'); % find edges of rotated image.

LJedge = bwlabel(Jedge); % labeled edges
%figure, imshow(label2rgb(LJedge)), title('all edges: labeled')
N= max(max(LJedge)); % number of labels
newLJedge= zeros(r,c); % Jmask is type double
m= 0;
for n= 1:N
 if sum(logical(sum(LJedge==n,2))) > 0.10*r; % if the edge spans >10%
of the whole height of the image, keep it. This filters out all of the
tiny edge segments and edge squiggles that are sometimes found with
the canny edge finding method.
 m= m+1;
 [x,y]= find(LJedge'==n);
 aveedge(m)= mean(x);
 stdedge(m)= std(x);
 if m>1
 deltaedge= aveedge(m)-aveedge(m-1);

 if deltaedge < mean([stdedge(m),stdedge(m-1)]) % if the m'th edge is
NOT significantly different from the m-1'th edge, make them one edge
 aveedge(m-1)= mean([aveedge(m-1),aveedge(m)]);
 stdedge(m-1)= mean([stdedge(m-1),stdedge(m)]);
 %disp([int2str(m),' th edge is not significantly different from the
previous edge.'])
 m= m-1;
 end

 end
 newLJedge(LJedge==n)= m; % place m in every pixel that contains a
contiguous edge; ie: rebuilding a label image
 end
end
%figure, imshow(label2rgb(newLJedge))

if m==1
 disp('Only found one edge, try analyzing a different ROI.')
elseif m>4
 disp('Found too many edges, try analyzing a different ROI.')
end


%% analyze only the outermost edge pixels
aveedge= [];
stdedge= [];
maxedgeroughness_pixels= [];
for mm=[1,m]
 if mm==1
 firstorlast= 'first'; % collect only left-hand pixels, removing all
duplicates to the right
 else % if mm==m
 firstorlast= 'last'; % collect only right-hand pixels, removing all
duplicates to the left
 end

 clear x y
 nn= 1;
 for n=1:r
 val= find(newLJedge(n,:)==mm,1,firstorlast);
 if ~isempty(val)
 x(nn)= val;
 y(nn)= n;
 nn= nn+1;
 end
 end

 aveedge= [aveedge, mean(x)];
 stdedge= [stdedge, std(x)];
 maxedgeroughness_pixels= max([maxedgeroughness_pixels, max(x)-
min(x)]);
 axes(handles.axes1);
 hold on, plot(x,y,'b'), hold off
end

avestripewidth_pixels= diff(aveedge);
rmsofedges_pixels= mean(stdedge);


%% output data
Stripe_Width_in_meters= avestripewidth_pixels/pixelspermeter
RMS_Edge_Roughness_in_meters= rmsofedges_pixels/pixelspermeter
Max_Edge_Roughness_in_meters= maxedgeroughness_pixels/pixelspermeter

axis off
set(handles.text5, 'String', num2str(Stripe_Width_in_meters,'%3.5g'));
set(handles.text13, 'String',
num2str(RMS_Edge_Roughness_in_meters,'%3.5g'));
set(handles.text15, 'String',
[num2str(Max_Edge_Roughness_in_meters,'%3.5g'), ' ± ',
num2str(1/pixelspermeter,'%3.5g') ]);



% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)
% hObject handle to FileMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject handle to OpenMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

global I filename filepath % pass in the previous filepath and pass
out the most current filepath

filepathtmp= filepath;
filenametmp= filename;
[filename,filepath,filenameindex]=
uigetfile({'*.bmp;*.gif;*.jpg;*.pbm;*.pcx;*.pgm;*.png;*.ppm;*.ras;*.ti
f','Image Files'},'Browse to Load an image; it need not be in the
origin directory.', filepath); % opens a typical windows file
selection panel and retrieves the selected filename, starting from the
last known filepath (which initializes to the pwd)
if filename~=0 % if a file was chosen
 if ~strcmp(filepathtmp,filepath)
 disp(['filepath= ',filepath])
 end
 disp(['filename= ',filename])
 I= imread([filepath,filename]);
 Ind= length(size(I)); % number of input image color dimensions
 if Ind==3
 I= rgb2gray(I); % this update supports color using standard weighted
sum= 0.2989*R + 0.5870*G + 0.1140*B, and supports high bit depth
images. Even if the input image is gray, but written in rgb mode, this
will still faithfully reproduce the gray coloring.
 elseif Ind==2
 % do nothing for grayscale images
 else
 filepath= filepathtmp;
 filename= filenametmp;
 disp('Filetype not supported, please convert to RGB or grayscale.')
 end
else % if a file was not chosen
 filepath= filepathtmp;
 filename= filenametmp;
 disp('Please load an image before proceeding.')
end



% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject handle to CloseMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
 ['Close ' get(handles.figure1,'Name') '...'],...
 'Yes','No','Yes');
if strcmp(selection,'No')
 return;
end

delete(handles.figure1)



% --- Executes on change in editbox edit2.
function edit2_Callback(hObject, eventdata, handles)
% hObject handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
% str2double(get(hObject,'String')) returns contents of edit2 as a
double

global pixelspermeter

pixelspermeter= str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
 set(hObject,'BackgroundColor','white');
end

global pixelspermeter

pixelspermeter= 311.5e6; % sets the default value when the GUI
launches
set(hObject, 'String', num2str(pixelspermeter,'%3.5g'));


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
OpenMenuItem_Callback(); % simply calls the filemenu open function


% --- Executes during object creation, after setting all properties.
function text5_CreateFcn(hObject, eventdata, handles)
% hObject handle to text5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

set(hObject, 'String', '');


% --- Executes during object creation, after setting all properties.
function text13_CreateFcn(hObject, eventdata, handles)
% hObject handle to text13 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

set(hObject, 'String', '');


% --- Executes during object creation, after setting all properties.
function text15_CreateFcn(hObject, eventdata, handles)
% hObject handle to text15 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

set(hObject, 'String', '');


% --- Executes during object creation, after setting all properties.
function uipanel4_CreateFcn(hObject, eventdata, handles)
% hObject handle to uipanel4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function uipanel7_CreateFcn(hObject, eventdata, handles)
% hObject handle to uipanel7 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called


% --- Executes during object creation, after setting all properties.
function uipanel8_CreateFcn(hObject, eventdata, handles)
```

```
% hObject handle to uipanel8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called


%% calibrate button
% --- Executes during object creation, after setting all properties.
function pushbutton4_CreateFcn(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called


% --- Executes during object deletion, before destroying properties.
function pushbutton4_DeleteFcn(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
pushbutton6.
function pushbutton6_ButtonDownFcn(hObject, eventdata, handles)
% hObject handle to pushbutton6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function pushbutton6_CreateFcn(hObject, eventdata, handles)
% hObject handle to pushbutton6 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called
```

List of Symbols, Abbreviations, and Acronyms

DARPA	Defense Advanced Research Projects Agency
e-beam	electron beam
GUI	graphical user interface
HSQ	hydrogen silsesquioxane
PEC	proximity effect correction
PSF	point spread function
RMS	root mean square
ROI	region of interest
SEM	scanning electron microscopy
SOI	silicon on insulator
UNIC	Ultraperformance Nanophotonic Intrachip Communication

NO. OF COPIES	ORGANIZATION
1 (PDF)	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP
7 (PDFS) 1 (HC)	US ARMY RSRCH LAB ATTN IMAL HRA MAIL & RECORDS MGMT (1 HC) ATTN RDRL CIO LL TECHL LIB ATTN RDRL SEE E J BICKFORD ATTN RDRL SEE E P PELLEGRINO ATTN RDRL SEE E N BAMBHA ATTN RDRL SEE E W CHANG ATTN RDRL SEE G WOOD ATTN RDRL SEE P GILLESPIE
1 (PDF)	DARPA ATTN MTO J SHAH